A distributed solution to K-out of- M resources allocation problem

Michel Raynal

UNITÉ DE RECHERCHE
INRIA-RENNES

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cedex
France
Tél. (1) 39 63 55 11

Rapports de Recherche

N° 1342

Programme 3
Réseaux et Systèmes Répartis

A DISTRIBUTED SOLUTION TO THE *K*-OUT OF-*M* RESOURCES ALLOCATION PROBLEM

Michel RAYNAL

Décembre 1990

RR-1342

IRISA INSTITUT DE RECHERCHE EN INFORMATIQUE ET SYSTEMES ALEATOIRES

Campus Universitaire de Beaulieu
35042 - RENNES CEDEX
FRANCE
Téléphone : 99.36.20.00
Télex : UNIRISA 950 473F
Télécopie : 99.38.38.32

A distributed solution to the k-out of- M resources allocation problem

Michel RAYNAL
IRISA
Campus de Beaulieu
35042 Rennes-cédex, FRANCE
raynal@irisa.fr

Programme 3

Publication Interne n° 559 - Novembre 1990 - 18 Pages

Abstract

We consider, in a distributed system, a set of M identical resources shared between n processes. Each of these resources can be used by at most one process at a given time (i.e. in mutual exclusion). In the k-out of-M resources allocation problem a process P_i can request at once any number k_i of these M resources ; this process remains blocked until it has got a set of k_i resources. A distributed algorithm, which generalizes the Ricart-Agrawala's mutual exclusion algorithm, is given for this problem ; a variant reducing the number of messages is also proposed. Finally this solution is extended to solve the generalized resources allocation problem in which a process request can concern several instances of different types of resources, each type being represented by some number of identical resources.

Index terms : K-out of-M resources allocation, distributed mutual exclusion, distributed synchronization, and-synchronization, permission-based algorithms.

Résumé

Une solution répartie au problème de l'allocation de ressources k parmi M

On s'intéresse à l'allocation de M ressources identiques dans un système réparti, l'utilisation de chacune d'elles étant soumise à la règle d'exclusion mutuelle. Dans l'allocation k parmi M un processus quelconque effectue ses demandes globalement pour un nombre quelconque k_i de celles-ci et reste bloqué jusqu'à les avoir obtenues. Un algorithme réparti, qui généralise l'algorithme d'exclusion de Ricart et Agrawala, est donné pour ce problème. Une variante, qui réduit le nombre de messages est également proposée. La solution est enfin étendue pour résoudre le problème de l'allocation généralisée de ressources dans lequel la demande d'un processus peut concerner plusieurs ressources de types différents, chacun représenté par un certain nombre de ressources identiques, la demande précisant le nombre d'exemplaires requis de chacun des types.

A distributed solution to the k-out of-M resources allocation problem

Michel RAYNAL
IRISA
Campus de Beaulieu
35042 Rennes-Cédex, FRANCE
raynal@irisa.fr

October 23, 1990

Abstract

We consider, in a distributed system, a set of M identical resources shared between n processes. Each of these resources can be used by at most one process at a given time (i.e. in mutual exclusion). In the k-out of-M resources allocation problem a process P_i can request at once any number k_i of these M resources ; this process remains blocked until it has got a set of k_i resources. A distributed algorithm, which generalizes the Ricart-Agrawala's mutual exclusion algorithm, is given for this problem ; a variant reducing the number of messages is also proposed. Finally this solution is extended to solve the generalized resources allocation problem in which a process request can concern several instances of different types of resources, each type being represented by some number of identical resources.

Index terms : k-out of-M resources allocation, distributed mutual exclusion, distributed synchronization, and-synchronization, permission-based algorithms.

1 The basic k-out of-M problem

We are interested in the management of a set of M identical resources shared by n processes $P_1, \ldots, P_i, \ldots, P_n$. At any time a resource can be used by at most one process, i.e. in mutual exclusion. On the other hand a process can request and use several of these resources at the same time. When a process wants to use any number k_i ($1 \leq k_i \leq M$) of these resources it requests them at once, and not one after the other, in order to avoid deadlock ; a process remains blocked until it has got the requested number of resources. After having used and released them it can make a new request for any number k'_i of these M resources. The k-out of-M problem lies in ensuring a correct management of these M resources, that is to say in the two following properties :

- safety property : the number of resources which are allocated to the processes at any time is always less than or equal to M. (Each resource being allocated to only one process at a time).

- liveness property : each request has to be satisfied within a finite time (under the hypothesis that the allocated resources are eventually released).

If we consider the case $M=1$, the k-out of-M problem reduces to the mutual exclusion one [10] ; if for any i, k_i is always equal to 1, the problem reduces to the multiple entries critical section one [1,2,9].

We consider the k-out of-M problem within the context of a distributed system composed of n sites ; we suppose one and only one process is associated to each site and we use the terms process and site without distinction. In such a system, the sites communicate only by exchanging messages along channels. These channels allow each process to send messages to each other ; they are reliable (no loss, no duplication, no spurious messages), and fifo (i.e. for each channel the delivery order of messages is the same as the sending one). Communication is asynchronous : transfer delays are finite but umpredictable.

2 Underlying ideas of the solution

In order to ensure the safety property a solution consists in giving a site P_i, that wants to use k_i resources, a consistent view of the number of resources used by the other sites. Let us call $used_i[j]$ the P_i's local view of the number of resources used by P_j. If $used_i[j]$ is greater than or equal to the number of resources actually used by P_j, then the test :

$$\sum_{1 \leq j \neq i \leq n} used_i[j] + k_i \leq M$$

is a consistent one ensuring safety. The management of these local variables has to be done accordingly.

One way to ensure liveness lies in putting a total order on the requests and serving them according to this order [5,11]. Such an order can easily be obtained by associating timestamps to requests (the timestamp attached to a request by a process has to be higher than any timestamp attached to a request received by this process ; timestamps can be generated using Lamport's rule [5]).

These two devices (over estimates of the number of resources used and total order on the requests) are combined to get the solution. Ricart-Agrawala's distributed mutual exclusion algorithm [12] is also based on these two principles. In this algorithm when a process P_i wants to enter the critical section it asks each other process the permission to enter ; it is only after having received the *(n-1)* permissions that a process can proceed (safety). Requests (asking the permissions) are timestamped and consequently totally ordered. This total order allows to ensure liveness : the sending of a permission by a requesting process is immediate or delayed according to the timestamps of the conflicting requests. As [9] we use this algorithm as a building block on which the solution to the k-out of-M problem is grafted. So the algorithm proposed belongs to the family of permission-based distributed algorithms [1,2,3,5,10,11,12,13].

3 The algorithm

Each site P_i is endowed with the following local context :

var booleen : $scdem_i$, ok_i, $prio_i$ **init** $false$;
integer : h_i, $maxh_i$ **init** 0;
[1..n] **of integer** : $used_i$ **init** 0;
set of 1..n : $delayed_i$ **init** ϕ;
integer: k_i;

The variable $scdem_i$ has the value *true* when P_i is requesting resources ; ok_i is true when P_i has got the resources it was waiting for ; $maxh_i$ represents the highest logical clock value ever received by P_i ; it allows it to timestamp its requests with h_i [5,12]. When P_i wants to use resources, it does not know how many resources are actually used by the other sites P_j. Such a P_j can be using an arbitrary number of them comprised between O and M ; consequently before sending its timestamped request to each site P_j, the process P_i increments by M each variable $used_i[j]$ in order its value be greater than or equal to the actual value.
When P_i receives from some P_j a request message *req(h,j)* (*(h,j)* is the timestamp of this request), it sends back a message *free(M)* if it is not interested in the resources or if its request has not priority over P_j's one (i.e. it has a higher timestamp). In the contrary case (P_i's request has priority over P_j's one) P_i sends back P_j a message $free(M - k_i)$ (where k_i is the number of resources P_i is requesting or using) ; this message indicate to P_j it can use only $M - k_i$ from resources P_i's point of view ; moreover P_i memorises, by including j into the set $delayed_i$, it will have to send P_j a message $free(k_i)$ when it has finished to use these k_i resources. In the particular situation where P_i is such that $j \in delayed_i$ when it receives a request from P_j, it can send back *free(M)* to P_j immediately : indeed in this cas P_j knows the number of resources P_i is using (P_j learnt this number previously with the P_i's answer to its preceding request ; moreover the P_i's request had priority over P_j's one).
The behaviour of each P_i is defined by the 4 following statements. Each of these statements is presented as a procedure body triggered by some event ; they are atomic, except for the one including a *wait* instruction. The order over the timestamps is defined as usual by

$$(h_i, i) < (h, j) \equiv (h_i < h \textbf{ or } (h_i = h \textbf{ and } i < j))$$

when requesting resources
begin $k_i :=$ *number of resources requested by* P_i;
$scdem_i := true$;
$ok_i := false$;
$h_i := maxh_i + 1$;
for $1 \leq j \neq i \leq n$ **do** $used_i[j] := used_i[j] + M$;
send $req(h_i, i)$ **to** P_j
od;
wait (ok_i);
end

when releasing the resources
begin $ok_i := false$;
for $j \in delayed_i$ **do send** $free\ (k_i)$ **to** P_j **od**;
$delayed_i := \phi$;
end

when receiving $req(h, j)$ **from** P_j
begin $maxh_i := max(maxh_i, h)$;
$prio_i := (scdem_i$ **or** $ok_i)$ **and** $(h_i, i) < (h, j)$;
case not $prio_i$: **send** $free(M)$ **to** P_j
$prio_i$: **if** $j \in delayed_i$ **then send** $free(M)$ **to** P_j $\quad (XX)$
else begin
$if\ k_i \neq M$ **then send** $free(M - k_i)$ **to** P_j **fsi**;
$delayed_i := delayed_i \cup \{j\}$
end
fi
endcase
end

when receiving free (y) **from** P_j
begin $used_i[j] := used_i[j] - y$;
if $scdem_i$ **and** $\sum_{x \neq i} used_i[x] + k_i \leq M$ **then** $scdem_i := false$;
$ok_i := true$ **fi**
end

The number of messages per use of a set of resources lies between *2(n-1)* and *3(n-1)*. The lower bound can be improved (cf. §5.1).

4 Proof

Proof of safety (respt. liveness) relies on the following propositions *P1* and *P2* (respt. *P4*).

4.1 Proposition *P1*

$$\forall i, j \ : \ used_i[j] \geq 0$$

The proof of this proposition is left to the reader.

4.2 Proposition *P2*

Let us consider the following situation. At a given time t the sites *i(1)*, ..., *i(g)*, ..., *i(m)* have carried out requests and are either waiting or using resources. These requests are ordered by their timestamps in the following way :

$$(h_1, i(1)) < (h_2, i(2)) < \ldots < (h_g, i(g)) < \ldots < (h_m, i(m))$$

Moreover the site *i(y)* is waiting for or using k_y resources (*i(m+1)* to *i(n)* are not requesting resources at time t).
If the condition allowing *i(g)* to use the requested resources is true then the condition is also true or will eventually be true for every site *i(x)* such that $1 \leq x < g$.

Proof

Let us examine the values the array $used_{i(g)}$ contain when *i(g)* evaluates to true its condition and the values the array $used_{i(x)}$ contains or will eventually contain (the word "eventually" is used to take into account the arbitrary transit delays of the messages along channels). The values are determined by the answers (messages *free*) of the sites to the requests of respectively *i(g)* and *i(x)*. The answer that some *i(y)* has sent for $used_{i(g)}[i(y)]$, and has sent or will eventually send for $used_{i(x)}[i(y)]$, has involved (or will involve) the following values :

i) $1 \leq y \leq x-1$:

$$used_{i(g)}[i(y)] = k_y$$
$$used_{i(x)}[i(y)] = k_y$$

ii) $x < y \leq n$:

ii1) value for *i(x)* :
the site *i(y)* is now either requesting with a higher timestamp (if $x+1 \leq y \leq m$) or not ($m+1 \leq x \leq n$). In either case *i(y)* might be requesting k'_y resources when it received the request from *i(x)*, with a request timestamp lower than the *i(x)* one. If it was the case we could have

$$used_{i(x)}[i(y)] = k'_y$$

but as the site *i(y)* has released its k'_y resources at the time t, the variable will eventually contain :

$$used_{i(x)}[i(y)] = 0$$

We get also this value if *i(y)* was not requesting when it receives the *i(x)* request.

ii2) for *i(g)* we have :

$$x \leq y \leq g-1 : \quad used_{i(g)}[i(y)] = k_y$$

$g < y \leq n$: the same reasonning as previously can be applied.

So by summing the values in each of the two arrays we can conclude :

if $(\sum_{y \neq g} used_{i(g)}[i(y)] + k_g)) \leq M$
then $(\sum_{y \neq x} used_{i(x)}[i(y)] + k_x)$is or will eventually be $\leq M$

In other words if the condition is true for *i(g)*, it is also true or will eventually become true (it depends on messages speed) for any site that has priority over *i(g)*.

Remark
This proposition clearly illustrates how the algorithm works. Requests are virtually satisfied in their timestamp order. The actual order depends on the message delays. If *M=1* actual and virtual orders are of course the same.

4.3 Safety *P3*

At most M resources can be used simultaneously (each resource being used by at most one process at a time).
Proof (by contradiction)
let us consider the situation depicted in proposition *P2* and suppose :

- that $\sum_{y<x} k_y \leq M$ (the requests of the processes *i(1)* to *i(x-1)* are or will be granted)
- and that, although the site *i(h)* ($x < h \leq g$) has evaluated its condition to true there are not enough resources to satisfy its request, i.e. : $\sum_{y \leq h} k_y > M$.

We have seen in *P2* first that when *i(h)* evaluates to true its condition we have :

$$\sum_{y \neq h} used_{i(h)}[i(y)] + k_h \leq M$$

and second that the left part of this inequation cannot be less than $\sum_{y \leq h} k_y$ (as *i(1)* to *i(x-1)* have not released their resources). Hence the contradiction : $M > M$, proving safety.

4.4 Proposition *P4*

Let us consider the following situation. There are K not granted resources and the unsatisfied request owning the lowest timestamp has been issued by *i(x)*, asks for $k_x \leq K$ resources and remains unsatisfied. We proof this situation cannot last indefinitely. In other words resources are allocated each time it is possible according to the timestamps of requests.

Proof
Let us consider the situation depicted in proposition *P2* with $1 \leq x \leq g$ and with :

$$\sum_{y<x} k_y + K = M$$

The answers to the *i(x)* request will eventually produce :

- $1 \leq y < x$: $used_{i(x)}[i(y)] = k_y$ after receiving the first *free* message from *i(y)*
  $= 0$ after *i(y)* released its resources.
- $x < y \leq n$: $used_{i(x)}[i(y)] = 0$.

So by summing we eventually obtain :

$$\sum_{y \neq x} used_{i(x)}[i(y)] + k_x \leq \sum_{y<x} k_y + k_x$$

as $k_x \leq K$ the condition for *i(x)* will then be true.

4.5 Liveness *P5*

Each request will be satisfied within finite time.
Proof
Let us consider the unsatisfied request with the lowest timestamp. If there are enough resources not granted to satisfy it, proposition *P4* applies. In the other case the requesting site will remain blocked until its condition becomes true. That request can be blocked only by requests endowed with a lower timestamp. When the sites that have issued these requests will release their resources, they will send *free* messages and as soon as enough *free* messages have been sent we are in the context of proposition *P4*. Consequently each request will be satisfied within a finite time.

5 Variants

5.1 Reducing the number of messages

It is possible to reduce the number of messages by taking into account the following fact. At the time P_i sends a request it knows already the number

of resources used by some P_j if $used_i[j] \neq 0$; so it is no use to send P_j a request in that case (consequently the statement noted XX in the algorithm can be removed). But it is now necessary for P_i to send P_j a request when it is waiting for resources and it receives from P_j a message *free(y)* such that $used_i[j]$ becomes *0* (in fact this value means either P_j dont use resources or P_i knows nothing about P_j). So an array $asked_i$ is used by each P_i to memorize the sites to which requests have been sent. (After P_i has sent its requests and is waiting, $used_i[j] = 0$ and $asked_i[j]$ to *true* mean that P_j do not use resources from P_i's point of view ; $asked_i[j]$ to *false* means P_i knows nothing about P_j and consequently has to ask P_j).
The modified algorithm is defined by the following behaviour for each P_i. The number of messages is now included between *0* and *3(n-1)* ; however the waiting time of a site can now be greater than in the first version as request messages can be sent later.

when requesting resources
begin $k_i := number\ of\ resources\ requested\ by\ P_i$;
$h_i := maxh_i + 1$;
for $1 \leq j \neq i \leq n$ **do**
if $used_i[j] = 0$ **then** $used_i[j] := M$;
$asked_i[j] := true$;
send $req(h_i, i)$ **to** P_j
else $asked_i[j] := false$
fi **od**;
$ok_i := \sum_{j \neq i} used_i[j] + k_i \leq M)$;
$scdem_i :=$ **not** ok_i;
wait (ok_i);
end

when releasing the resources
begin $ok_i := false$;
for $j \in delayed_i$ **do send** $free(k_i)$ **to** P_j **od**;
$delayed_i := \phi$;
end

when receiving $req(h, j)$ **from** P_j

```
begin maxh_i := max(maxh_i, h);
      prio_i := (scdem_i or ok_i) and ((h_i, i) < (h, j));
      case not prio_i : send free(M) to P_j
           prio_i :     begin
                        if k_i ≠ M then send free(M − k_i) to P_j fi;
                        delayed_i := delayed_i ∪ {j}
                        end
      endcase;
end

when receiving free(y) from P_j
begin used_i[j] := used_i[j] − y;
      if scdem_i and used_i[j] = 0 and not asked_i[j]
         then used_i[j] := M;
              send req(h_i, i) to P_j;
              asked_i[j] = true
      fi;
      if scdem_i and (Σ_{x≠i} used_i[x] + k_i ≤ M) then scdem_i := false;
                                                   ok_i := true   fi;
end
```

5.2 Adaptation to others topologies

A fully connected network has been implicitly assumed to carry *req* and *free* messages. The algorithm can be adapted to take into account a ring topology by using the principle described in §6.3. of [12], or a tree topology [8,14] or an arbitrary network [4].

6 A solution to the generalized AND-allocation

Now we consider there exist several types of resources : *R(1), ..., R(p)*. Each type is represented by several identical resources : there is $M(\alpha)$ instances of the resource type $R(\alpha)$. A process P_i requests simultaneously all the

resources it needs ; in the following demand for example :

$$demand((k_i(\alpha), R(\alpha)), (k_i(\beta), R(\beta))) \quad (Z)$$

the process P_i asks for $k_i(\alpha)$ resources of type $R(\alpha)$ and $k_i(\beta)$ resources of type $R(\beta)$. P_i remains blocked until it has obtained all the requested resources. This problem is a generalized AND-allocation one (several types of resources are requested and for a given type $R(\alpha)$ the request asks for $k_i(\alpha)$-out of-$M(\alpha)$ resources).
The preceding algorithm constitutes a basic building block to obtain a simple solution to this problem. Let us call $GR(\alpha)$ the set of sites which share the $M(\alpha)$ resources of type $R(\alpha)$; in such a set the sites have distinct identities. Giving a unique timestamp to each demand allow to solve all the conflicts in the same way whatever is (or are) the resource(s) concerned. So, for example, if P_i and P_j are in conflict to use resources of type $R(\alpha)$, $R(\beta)$ and $R(\gamma)$, these three conflicts are solved in the same way : either in favour of P_i or P_j according to the two timestamps of their requests.
As an illustration, here is the program text relative to the demand by P_i described in line (Z) ; it concerns 2 types of resources. The text is obtained from the initial version (the other texts are obtained in the same way).

when requesting resources
begin
$k_i(\alpha) :=$ *number of resources of type* $R(\alpha)$ P_i *needs*;
$k_i(\beta) :=$ *number of resources of type* $R(\beta)$ P_i *needs*;
$h_i := maxh_i + 1$;
$scdem_i(\alpha) := true$; $ok_i(\alpha) := false$;
$scdem_i(\beta) := true$; $ok_i(\beta) := false$;
for $x = \alpha, \beta$
do $\%used_i(x)$ *is the array relative to the resource type* $R(x)$; *its entries are the elements of* $GR(x)\%$
for $j \in GR(x), j \neq i$
do $used_i(x)[j] := used_i(x)[j] + M(x)$;
send $req(R(x), (h_i, i))$ **to** P_j
od
od;
wait ($ok_i(\alpha)$ **and** $ok_i(\beta)$);
end

Messages *req* and *free* take the resource type concerned as a parameter. Statements associated to the reception of these messages use the data structures associated to the type of resources defined by the corresponding parameter.

Remark

The drinking philosophers problem [3] is some kind of AND-allocation problem. Each philosopher (i.e. site or process) shares bottles with it neighbhours and can request any subset of these ones. However between two neighbhours philosophers sharing M bottles, neither philosopher can request k out of M of these bottles : his request must concern an a priori defined set of k bottles and not an arbitrary set of k of these M bottles (yet he may need distinct sets of bottles for different consecutive requests). A very interesting property of the Chandy-Misra's solution to the drinkers problem lies in using only bounded variables (there are no timestamps).

7 Conclusion

The proposed solutions to solve the basic and the generalized resources allocation problems are simple. They have been designed from a well-known building block, namely the Ricart-Agrawala's distributed mutual exclusion algorithm [12]. Other basic building blocks are possible : algorithms based on a travelling token [6,7,8], or algorithms using a more sophisticated permission-based protocols [11,13]. An interest on the proposed solution lies in its simplicity. The variant illustrated some tradeoff between response time and number of messages.

8 References

References

[1] CARVALHO O.S.F., ROUCAIROL G. *Assertion decomposition and partial correctness of distributed algorithms.* In Distributed Computing, (Paker and Verjus Ed.), Academic Press, (1983), pp. 67-93.

[2] CARVALHO O.S.F., ROUCAIROL G. *On the distribution of an assertion.* Proc. of the 2d ACM Sigact-Sigops Symposium on P.O.D.C., (August 1982), pp. 121-131.

[3] CHANDY K.M., MISRA J. *The drinking philosophers problem.* ACM Transactions on Programming Languages and Systems, 6(4) :632-646, (1984).

[4] HELARY J.M., PLOUZEAU N., RAYNAL M. *A distributed algorithm for mutual exclusion in an arbitrary network.* The Computer Journal, 31(4) : 289-295, (1988).

[5] LAMPORT L. *Time, clocks and ordering of events in distributed systems.* communications of the ACM, 21(7) : 558-564, (1978).

[6] LE LANN G. *Distributed systems : towards a formal approach.* IFIP congress, Toronto, (1977), pp. 155-160.

[7] MARTIN A.J. *Distributed mutual exclusion on a ring of processes.* Science of Computer Programming, vol.5, (1985), pp. 265-276.

[8] RAYMOND K. *A tree-based algorithm for distributed mutual exclusion.* ACM Transactions on Computer Systems, 7(1) : 61-77, (1989).

[9] RAYMOND K. *a distributed algorithm for multiple entries to a critical section.* Inf. Processing Letters, vol.30, (1989), pp. 189-193.

[10] RAYNAL M. *Algorithms for mutual exclusion.* MIT Press, (1986), 107 p.

[11] RAYNAL M. *Distributed computations and networks : concepts, tools and algorithms.* MIT Press, (1988), 160 p.

[12] RICART G., AGRAWALA A.K. *an optimal algorithm for mutual exclusion in computer networks.* Communication of the ACM, 24(1) : 9-17, (1981).

[13] SANDERS B. *The information structure of distributed mutual exclusion algorithms.* ACM Transactions on Computer Systems, 5(3) : 284-299, (1987).

[14] Van De SNEPSCHEUT J.L. *Fair mutual exclusion on a graph of processes.* Distributed Computing, 2 : 113-115, (1987).

LISTE DES DERNIERES PUBLICATIONS INTERNES IRISA

PI 548 LES PREDICATS COLLECTIFS : UN MOYEN D'EXPRESSION DU CONTROLE DU PARALLELISME "OU" EN PROLOG
René QUINIOU, Laurent TRILLING
Septembre 1990, 34 Pages.

PI 549 NORMALISATION SOUS HYPOTHESE D'ABSENCE DE LIEN APPLICATION AU CAS NOMINAL
François DAUDE
Septembre 1990, 42 Pages.

PI 550 MULTISCALE SIGNAL PROCESSING : FROM QMF TO WAVELETS
Albert BENVENISTE
Septembre 1990, 28 Pages.

PI 551 ON THE TRANSITION GRAPHS OF AUTOMATA AND GRAMMARS
Didier CAUCAL, Roland MONFORT
Septembre 1990, 46 Pages.

PI 552 ERREURS DE CALCUL DES ORDINATEURS
Jocelyne ERHEL
Septembre 1990, 58 Pages.

PI 553 SEQUENTIAL FUNCTIONS
Boubakar GAMATIE, Octobre 1990, 16 Pages.

PI 554 ANALYSE DE LA FORME D'UN COEFFICIENT D'ASSOCIATION ENTRE VARIABLES QUALITATIVES
Mohamed OUALI ALLAH
Octobre 1990, 26 Pages.

PI 555 APPROXIMATION BY NONLINEAR WAVELET NETWORKS
Qinghua ZHANG, Albert BENVENISTE
Octobre 1990, 16 Pages.

PI 556 CONCEPTION ET INTEGRATION D'UN CORRELATEUR SYSTOLIQUE
Catherine DEZAN, Eric GAUTRIN, Patrice QUINTON
Novembre 1990, 16 Pages.

PI 557 VARIATIONAL APPROACH OF A MAGNETIC SHAPING PROBLEM
Michel CROUZEIX
Novembre 1990, 14 Pages.

PI 558 THE DAVIDSON METHOD
Michel CROUZEIX, Bernard PHILIPPE et Miloud SADKANE
Novembre 1990, 22 Pages.

PI 559 A DISTRIBUTED SOLUTION TO THE k-OUT OF-M RESOURCES ALLOCATION PROBLEM
Michel RAYNAL
Novembre 1990, 18 Pages.

PI 560 A SIMPLE TAXONOMY FOR DISTRIBUTED MUTUAL EXCLUSION ALGORITHMS
Michel RAYNAL
Novembre 1990, 8 Pages.

ISSN 0249 - 6399